9 CLAIMS

I claim:

1. A novel programming grammar for building regular expressions, including regular expression forms not found in the art.

2. A novel programming grammar for building regular expressions which models C-style languages in terms of statement blocks, function blocks, scoped variables and their declarations, expression binding rules, primitive data types, and array declarations.

3. The novel programming grammar of claim 2, wherein the composition of a regular expression uses the standard expression operators and expression binding rules of the C-language such that a regular expression consists of compositions using unary and binary operators of the C-language following exactly the C-model of associativity and precedence for borrowed/overloaded operators, comprising literals and variables as with C-expressions, and further comprising new operators unfound in C-like languages but which follow the C-rules of associativity and precedence, and allowing for the novel regular expression forms of this grammar.

4. The novel programming grammar of claim 3, wherein the grammar further comprises C/C++ expression syntax for building regular expressions wherein existing C-style unary and binary operators are overloaded to form certain regular expression compositions, wherein existing binary operators are used as unary operators, and wherein new operators are created, in all cases observing the C-language rules of associativity and precedence of operators.

5. The novel programming grammar of claim 3, wherein the grammar further comprises a novel primitive data type of the language called Pattern, an immutable Object in the same sense as a Java String, which serves to hold **ANY** regular expression form available in the grammar.

6. The novel programming grammar of claim 3, wherein a union composition regular expression adopts the standard C++ "|" (vertical bar) operator in the normal position in the precedence table of operators, and carries the normal left-to-right associativity for binding in the absence of parentheses.

7. The novel programming grammar of claim 3, wherein a normal (greedy from left to right) concatenation regular expression composition uses the standard C++ "+" (plus sign) operator, chosen to emulate Java string concatenation which is also accomplished by the + operator.

8. The novel programming grammar of claim 3, wherein a special (greedy from right-to-left) regular expression form of concatenation regular expression composition uses the invented operator "<+" (less than plus sign) in the C++ precedence table at the same position as the + operator and wherein the associativity of concatenation and the choice of right-to-left or left-to-right operators are significant.

9. The novel programming grammar of claim 3, wherein a normal (greedy from left-to-right) regular expression concatenation also uses the invented operator ">+" (greater than plus sign) in the C++ precedence table at the same position as the + operator and with exactly the same meaning and usage as the + operator.

10. The novel programming grammar of claim 3, wherein the C++ binary "*" (asterisk) operator is used to create regular expressions of the composition semantics "repeat a given regex N or more times", wherein the composition is of the form - Pattern * integer - and wherein this operator binds at the normal position of the * operator in the C precedence table.

11. The novel programing grammar of claim 3, wherein the C++ binary "*" (asterisk) operator is used to create regular expressions of the composition semantics "iterate a given regex from N1 to N2 times, inclusive", - as in Pattern * Range - , wherein the Range data type is new built-in data type of the language and is accomplished by the expression - int .. int, wherein ".." (dot-dot) is a special operator of this grammar that binds tightest of any operator in the C-style precedence table of this grammar.

12. The novel programing grammar of claim 3, wherein the C++ binary "*" (asterisk) operator is used in the unary sense to compose regular expressions of the composition semantics "repeat a given regex 0 or more times" as in - * Pattern - , wherein the unary repeat operator must be placed before the pattern in order to fit into the C-expression model.

13. The novel programming grammar of claim 3, wherein the "+" (plus sign) operator is used in the unary sense to compose regular expressions of the composition semantics "repeat a given regex 1 or more times" as in - + Pattern - , wherein the unary repeat operator must be placed before the pattern in order to fit into the C-expression model.

14. The novel programming grammar of claim 3, wherein the "?" (question mark) metacharacter-operator is used to optionalize a regular expression, wherein the optionalize-grammar uses this operator preceding the pattern, and wherein the operator-metacharacter has a unary purpose in addition to its standard binary purpose.

15. The novel programming grammar of claim 3, wherein the conditional regular expression uses the "**?**" (question mark) operator as in - bool ? Pattern1 : Pattern2 - as a natural by-product of how the C-interpreter handles the conditional operator.

16. The novel programming grammar of claims 1 or 3, wherein arbitrary instructional side-effects are embedded directly into regular expressions through the novel do-pattern form and through other novel composition forms which are implemented in whole or in part via implicit do-patterns.

17. The novel programming grammar of claim 16, wherein the syntax of the explicit do-pattern comprises -do (preStmt1, preStmt2, . . ., preStmtN1; embed-regex; postStmt1, postStmt2, . . ., postStmtN2)- which is modeled closely after the C-style for-statement, wherein comma separated statements are embedded before and after the loop continuation expression, "do" is substituted for "for", and the loop-continuation-expression is replaced with a regular expression.

18. The novel programming grammar of claim 3, further comprising a CapturePattern syntax wherein a regular expression has its match results captured into a named variable of any scope available to the CapturePattern, the named variable consisting of a global variable, a function parameter, a production rule parameter, a variable declared in a function block, or a statement block of the function, or a variable declared within the scope of a DoPattern, and wherein the grammar is - &myvar (reg-ex) -.

19. The novel programming grammar of claim 16, wherein instantiated production rules generate implicit side-effects for proper binding of the in-out parameters and out parameters of the reference to the actual variable being passed.

20. The novel programming grammar of claim 16, wherein the side effects of any composition can be nullified by the novel inhibit grammar comprising - inhibit (regex) -.

21. The novel programming grammar of claims 1 or 3, wherein a subjunctive grammar is used to qualify or reject matches to the primary via the secondary wherein any side-effects embedded in the secondary are inhibited during qualification/rejection, and wherein any side-effects embedded in the primary are not affected by qualification/rejection since if qualification occurs, the expression executes exactly as if the subjunctive expression had been replaced by the primary and if rejection occurs, there is no match and no candidate for a winning thread, hence no instructional side-effects.

22. The novel programming grammar of claims 1 or 3, further comprising means for the programmer to control side- effects actually triggered as a result of matching a regular expression to a stream/string without ambiguity.

23. The novel programming grammar of claim 22, wherein means for the programmer to control side- effects actually triggered as a result of matching a regular expression to a stream/string without ambiguity further comprises instruction-arcs which represent all side-effects in the automata, which do not contribute to character-matching characteristics of any automata thread, and which therefore do not affect the accept/reject outcome of any composition or sub-composition automata.

24. The novel programming grammar of claim 23, wherein the instruction arcs do affect the arcnum-sequence priority given to arcs which helps select a winning automata execution thread to resolve ambiguity.

25. The novel programming grammar of claim 23, wherein adding or removing side-effects to/from portions of a regular expression does not affect whether or not any other side-effects will be part of the winning thread.

26. The novel programming grammar of claim 23, wherein if side-effects from a sub-expression of a regular expression are part of a winning thread, than any outer low-level implicit or explicit do-patterns which also contain the side-effects will also have their side-effects included on the winning path, in proper nested order.

27. The novel programming grammar of claim 23, wherein upon completion of automata execution and acceptance of the input string, all side-effects triggered for post-acceptance execution reside in one and only one coherent path through the composition automata, providing at most one set of side-effects fired for the execution of a given automata.

28. The novel programming grammar of claim 23, wherein all rejected candidate paths through the automata which are rejected at any stage of automata execution thread pruning or during subset construction can only be rejected by the existence of a superior path through the graph that recognizes the same characters according to the ambiguity priority rules of the unary and binary operators of the grammar, and wherein binary rules can be described as - P = Pl op P2 - .

29. The novel programming grammar of claim 23, wherein in terms of left-to-right concatenation, the left term is to be greedier than the right term wherein if the composition or sub-composition recognizes the same set of characters in multiple allocations to the two terms, the winning automata thread will select the allocation of the fewest characters to the right term, and which is significant only when the concatenation composition contains side-effects.

30. The novel programming grammar of claim 23, wherein in terms of right-to-left concatenation, the right term is to be greedier than the left term wherein if the composition or sub-composition recognizes the same set of characters in multiple allocations to the two terms, the winning automata thread will select the allocation of the fewest characters to the left term, and which is significant only when the concatenation composition contains side-effects.

31. The novel programming grammar of claim 23, wherein for union when both terms match exactly the same number of characters, the right term is always preferable to the left term, and which is significant only when the concatenation composition contains side-effects.

32. The novel programming grammar of claim 31, wherein for union when the terms match a different number of characters, the greedier term is preferred as a result of higher importance (of greediness) than left-to-right or top-to-bottom precedence in the union.

33. The novel programming grammar of claim 23, wherein for repeat or iterate compositions defined by - P = P1 op N, where N is either the minimum number of iterations expected or the iteration from-to-range, ambiguity resolution rules provide that no losing path through the sub-expression P will match a greater number of characters on its first match to P1, and likewise in turn on each subsequent match, and which is of possible significance in the presence of side-effects within P1.

34. The novel programming grammar of claim 33, wherein ambiguity resolution rules are covered for unary repeats in that the unary repeats can be converted to binary repeats.

35. The novel programming grammar of claim 23, wherein ambiguity resolution rules proceed from higher composition levels to lower levels wherein the optimal thread is chosen based upon a mismatch between two threads at the highest level of composition and is justified by the ambiguity resolution rules involving sequencing of arcnum sequences, and wherein (by example) inner repeats are as greedy as possible relative to outer repeats (relative to side-effects), and wherein the associativity binding of a concatenation of 3 or more terms can be significant in the presence of side effects in any of those terms.

36. The novel programming grammar of claim 23, wherein ambiguity resolution rules provide that when matching a stream against a pattern, after all the rules have fired, and two or more paths are presented each of which eats a different number of characters, the path/subpath which eats the greater number of characters is given preference.

37. The novel programming grammar of claims 1 or 3, wherein the production rules are part of the language at any scope where a function can be defined, wherein production rules can be parameterized in the same form and syntax of param-decl-lists of functions, wherein production rules are instantiated (as templates) rather than called, and wherein the syntax for parameterization of a production rule's embedded regular expression is - production MyRule <param-decl-list> reg-ex ; -.

38. The novel programming grammar of claim 37, wherein parameterization of a production rule's embedded regular expression allows the recognition characteristics of the instantiated rule to vary through in/default parameters to create reusable design patterns.

39. The novel programming grammar of claim 37, wherein parameterization of a production rule's embedded regular expression allows the side-effect characteristics of the instantiated rule to vary through in out, out, or default in container parameters (such as arrays) to create a rule that can be instantiated to capture into or modify parameters rather than actual variables.

40. The novel programming grammar of claim 21, wherein qualification/rejection of matches by a subjunctive regular expression involves a primary regular expression and a secondary regular expression and an automata composition engine which uses same-time arrival at the exit nodes of both the primary regular expression and the secondary regular expression to determine qualification/rejection.

41. The novel programming grammar of claim 40, wherein in the case of the subjunctive ButGrammar a match to the primary regular expression is accepted only if the secondary regular expression also has a same-time arrival to its exit node, wherein the ButGrammar is expressed as - primary-regex but qualifier-regex -, wherein primary-regex and qualifier-regex are any regular expression possible in the language, and wherein instructional side-effects of the secondary/qualifier regular expression are automatically nullified/inhibited, and all instructional side-effects of the primary regular expression are preserved in order, so long as they fall on a winning thread which has not been disqualified by the qualifier regular expression.

42. The novel programming grammar of claim 40, wherein in the case of the subjunctive ButnotGrammar a match to the primary regular expression is rejected if the secondary regular expression also has a same-time arrival to its exit node, wherein the ButnotGrammar is expressed as - primary-regex butnot qualifier-regex -, wherein primary-regex and qualifier-regex are any regular expression possible in the language, and wherein instructional side-effects of the secondary/qualifier regular expression are automatically nullified/inhibited and all instructional side-effects of the primary regular expression are preserved in order, so long as they fall on a winning thread which has not been disqualified by the qualifier regular expression.

43. The novel programming grammar of claim 40, wherein subjunctive chaining is allowed (with left-to-right associativity), and wherein by example the following expression - primary-regex but qualifier-regex1 but qualifier-regex2 - is implied, and wherein primary-regex but qualifier-regex1 is itself a primary regular expression and is then subjuncted with qualifier-regex2, as the secondary [qualifier] regular expression.

44. The novel programming grammar of claims 1 or 28, wherein automata execution-time thread pruning is achieved by limiting no more than one thread to be at the same node at the same time at any character position snapshot of the execution, with the winning thread determined by comparing the sequences of arcnum-sequnces traversed to that point, the algorithm of which ensures that no regular expession in the grammar can produce an automata which will cause the engine to require exponential time (in the size of the input stream) to execute, with expected linear-time execution and worst-case N-squared execution (in the size of the input stream), wherein exponential behavior of certain capturing non-deterministic finite automata of other grammars, such as nested indeterminate repeat expressions, is eliminated in equalivalent expressions of the grammar.

45. The novel programming grammar of claim 37, wherein proper binding (when instantiated) of production rule templates is accomplished by a novel runtime virtual machine abstraction nominated by the author as a template frame.

46. A method of practicing the novel programming grammar of claim 40, the method comprising the step of using the subjunctive to enhance expressivity in lieu of using complex compositions involving negated character-classes.

47. The method of claim 46, the method further enhancing expressivity and further comprising the steps of integrating side-effects into the primary expressions of a subjunctive grammar, observing the existing ambiguity resolution rules of the primary expression according to the grammar as in claims 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, or 36, and using the subjunctive to produce resulting graphs which are not as well expressed (or at all) without the subjunctive in the presence of full ambiguity resolution.

48. A method of practicing the novel programming grammar of claim 5 for step-by-step compositions, the method comprising the step of using the pattern data type immutability of the grammar wherein once a Pattern object is instantiated and assigned to a variable, it never will change, even if the variable is reassigned.

49. A method of practicing the novel programming grammar of claims 1 or 3, the method comprising the step of using the production rules of the grammar which can be parameterized wherein a reusable design pattern need be coded only once as a production rule template, or a few times, as a set of related templates with the same name but different signatures, wherein the reusability of the rule allows parameterization of both the recognition portions of the resulting graph (as with passing another regex as an in parameterization) as well as parameterization of the side-effects through out and in out params or with an in parameterization consisting of a container structure such as an array.

50. A method of practicing the novel programming grammar of claim 3, the method comprising the step of using the reject literal of the grammar to compose dynamic unions wherein the elements of the union are pulled from container objects, such as arrays and the reject provides the union a valid initial value.

51. The novel programming grammar of claim 1, wherein the grammar further comprises parameterizable production rules wherein the programmer can abstract both recognition and side-effect characteristics of a useful regular expression.

52. The novel programming grammar of claim 3, wherein the grammar further comprises parameterizable production rules wherein the programmer can abstract both recognition and side-effect characteristics of a useful regular expression.

53. The novel programming grammar of claim 1, wherein the grammar further comprises a design pattern for extracting complex structures via nested do-patterns and capture patterns.

54. The novel programming grammar of claim 3, wherein the grammar further comprises a design pattern for extracting complex structures via nested do-patterns and capture patterns.

55. The novel programming grammar of claim 35, wherein a production rule can bind an in-parameterization to itself during instantiation, allowing the rule-body to include do-patterns which manipulate the in-parameterization, such that side-effect statements which reference the in-parameter (within the body-expression of the production rule) are referencing the value at time of instantiation rather than the value at the time of tokenize-statement, a type of "unpinning" which fits well within the goal of step-by-step composition.

56. The novel programming grammar of claim 1, wherein the grammar further comprises a design pattern making a Pattern/regular expression an “in” parameterization of rule, wherein other regular expression instantiations are passed during instantiation yielding a production rule as a formula for building a regular expression (from other regular expressions) rather than the regular expression itself.

57. The novel programming grammar of claim 3, wherein the grammar further comprises a design pattern making a Pattern/regular expression an “in” parameterization of rule, wherein other regular expression instantiations are passed during instantiation yielding a production rule as a formula for building a regular expression (from other regular expressions) rather than the regular expression itself.